REMARKS

Claims 1-24 and 26 are all the claims pending in the application.

Claims 8 and 20 have been rejected under § 112 (second paragraph) as being invented for essentially being broader than the claims from which they depend. These claims have been cancelled herein, without prejudice.

Claims 1, 3, 5-13, 15, 17-24 and 26 are rejected under § 103(a) as being unpatentable over Nori, et al. (WO 01/014962) in view of Kohl (U.S. Patent No. 6,163,878). Still further, claims 2, 4, 14, and 16 are rejected under § 103(a) as being unpatentable over Nori in view of Kohl, and further in view of Iriuchijima, et al. (U.S. Patent No. 6,070,006).

Applicant has canceled dependent claims 6, 8-11, 18, and 20-23, leaving claims 1-5, 7, 12-17, 19, 24 and 26 pending in the application. It is believed that these claims are patentable over the prior art for the following reasons.

The Examiner argues in the Office Action that he has considered Applicant's arguments filed earlier, and is still of the opinion that the independent claims 1, 13 and 26 are obvious over Nori in view of Kohl.

As admitted by the Examiner in earlier Office Actions, Nori fails to disclose the features b) and c) of the independent claims 1, 13 and 26.

We strongly disagree with the interpretation of obviousness of the Examiner relating to the Kohl reference, as it is believed that the Examiner at least uses hindsight to make his arguments.

For example, the examiner argues that Kohl (column 7, line 39 — column 8, line 10) teaches that "users ... may specify specific 'pages' of the application that may be presented to

various groups and/or users" using an "Access Control List ... [which] is used to determine the application data being retrieved from the ADB..."

Respectfully, this does only express the fact that the application developer can define permissions of a certain *(group of) user(s)* in relation to a certain class of data in the ACL. It does NOT express the fact that actual relations in the live application data can be used to define the actual permissions at runtime. See also Kohl — Fig. 4B where the use of the ACL is shown.

Applicant respectfully submits that the mechanism taught by Kohl (column 7 , line 39 — column 8, line 10) only uses static relationships between user (groups) and classes (the application's data *model*), and is thus significantly less advanced compared to the mechanism of the own method for access permissions, as described and claimed by applicant.

Applicant submits that the **own** method allows the application developer to grant access permissions for any user (group) to any part of the application data in such a way that access to an *individual* data object depends on the relationship in the live application data of that particular object to the current user.

Furthermore, applicant submits that the **follow foreign object** (f.f.o.) method allows the application developer to grant access permissions for any user (group) to any part of the application data in such a way that access to an individual data object depends on *another* individual data object when the latter object has a foreign relationship with former object. This means that the access permissions on a certain object are propagated to related objects. See the example below.

Applicant respectfully argues that the f.f.o. mechanism is significantly more advanced compared to the mechanism taught by Kohl (see e.g. column 7, line 39 — column 8, line 10).

For the examiner's convenience, the following pages contain an elaboration on the example given in page 30, line 17 — page 34, line 13 of the PCT publication -W02004/021179 of applicant, corresponding to paragraphs [0235]-[0265] of the US patent publication US2006/117294 of applicant, illustrating the principle of the own and **f.f.o.** methods.

In this example we have 3 relevant classes: Persons, Homes and Rentals.

Suppose in the running application data we have (in runtime):

- 2 objects of class *Person*: person P1 and person P2, which also represent users of the application. P1 is Owner, while P2 is an Agent;
- 2 objects of class *Home*: home H1 and home H2; and
- 2 objects of class *Rental*: rental R1 and rental R2.

Also in the running application data relations exists between these objects (in runtime):

- home H1 is related to rental R1 and person P1;
- home H2 is related to rental R2 and person P2..

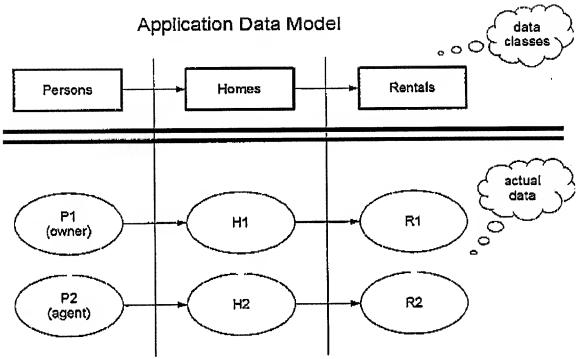

Figure 1 - Example application

Now, let's consider the given permission matrix (page 32, line 22). For the examiner's convenience, here is an excerpt from the given permission table.

	Homes					Rentals				
	Insert	*Delete*	*Select*	*Read*	*Update*	*Insert*	*Delete*	*Select*	*Read*	*Update*
Owner	O	O	Y	C	O	N	N	O	F	N
Agent	O	O	Y	C	O	O	O	O	F	O

Legend:

Code	Access
Y	Yes
N	No
C	Conditional
O	Own
F	Follow foreign object

Table 1 – Excerpt from permission table in page 32, line 22

From the matrix we see that the update permission for Owners in relation to Homes is: **Own**. This means that owners are only allowed to update their own data. And although they might be allowed to read data about other homes, they are only allowed to update their own homes. Owners are not allowed to update data about the other homes.

In this example: User P1 is only allowed to update HI, not H2. The permissions are defined runtime from the actual application data. In no way, Kohl teaches a mechanism where actual application data is used to determine the access permissions.

Now, let's have another look at the same permission matrix. From the matrix we see that the read permission for Persons (with role Owner) in relation to rentals is: **Follow foreign object.** This means that owners are only allowed to read data about a specific rental when they are allowed to read data about the foreign object. In this example the foreign object is an object of class Home. So, when a Person is allowed to read data about a certain Home, this person is also allowed to read the data about the related Rental(s). When a Person is NOT allowed to read data about a certain Home, this person is also not allowed to read the related Rental(s). In this

example, when a user is allowed to read HI, then the user is also allowed to read Rl. When the user is not allowed to read H1, this user is also not allowed to read Rl.

The big advantage of this mechanism is that it lowers the complexity of application development. Suppose the read permission related to Homes is the result of a complex calculation. In any conventional application development environment, the calculation of the read permission of the Rentals will be as complex as the calculation of the read permission of the Home. In the current invention, this is accomplished by simply setting the permission to 'Follow foreign object'.

Again, to summarize, in no way does Kohl reveal a mechanism like this.

As should be appreciated from the foregoing, the skilled person would never find the solution according to the present claims 1, 13 and 26 solely from combining the teachings of Nori and Kohl with any reasonable expectation of success. Only by exercising an inventive skill, the combination of features of the present independent claims 1, 13 and 26 will be discovered.

In the above, Applicant has clearly demonstrated that both the 'own' and 'follow foreign object' mechanism are novel and inventive over the prior art.

In view of the above, reconsideration and allowance of this application are now believed to be in order, and such actions are hereby solicited.

If any points remain in issue which the Examiner feels may be best resolved through a personal or telephone interview, the Examiner is kindly requested to contact the undersigned at the telephone number listed below.

The USPTO is directed and authorized to charge all required fees, except for the Issue Fee and the Publication Fee, to Deposit Account No. 19-4880. Please also credit any overpayments to said Deposit Account.

Respectfully submitted,

/Brian W. Hannon/

Brian W. Hannon
Registration No. 32,778

SUGHRUE MION, PLLC
Telephone: (202) 293-7060
Facsimile: (202) 293-7860
WASHINGTON OFFICE
23373
CUSTOMER NUMBER

Date: July 28, 2010